Stat 274 Assignment 2

Karl Jiang

October 23, 2016

1)

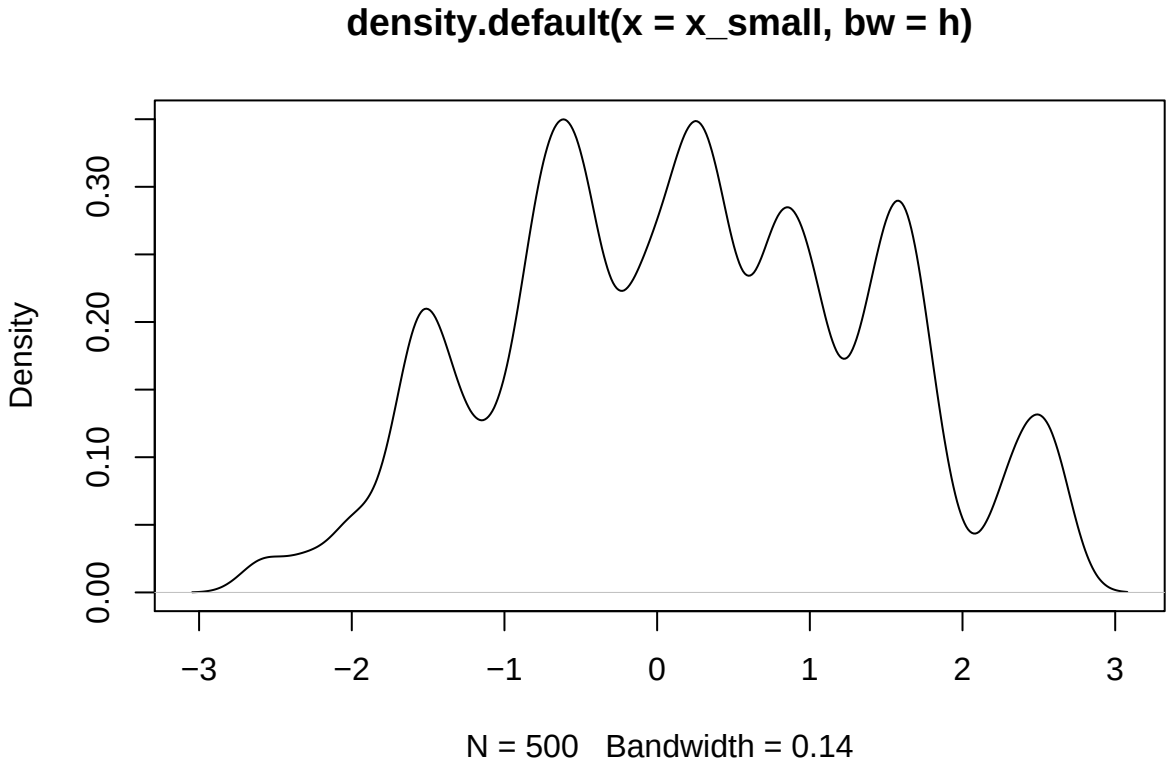
b)

Take smoothed density. Large bandwidth. Below is my implementation for the meanshift algorithm

```
#x is a vector of float, assumes gaussian kernel, w is point
#if start point is 0/FALSE, then it'll generate a random point to start.
#bw.ucv
ms_step <- function(w, x, h) {
  weighted <- sum(x * dnorm((w - x) / h) )
  total_weight <- sum(dnorm((w - x) / h) )
  return(weighted / total_weight)
}

#the actual function. Goes until the diff in w_t+1 and w is < tol.stop
est_mode <- function(x, h, tol.stop = 1e-06, start_point = 0) {
  w <- start_point
  if(!start_point){
    w <- runif(1, min(x), max(x))
  }

  w_next <- ms_step(w, x, h)
  while(abs(w_next - w) > tol.stop) {
    w <- w_next
    w_next <- ms_step(w, x, h)
  }
  return(w_next) #mode
}
```

Using it on the 500 length vector. Note that I choose the bandwidth h to be 0.14, because even though it may not be best performer under cv, it makes the density have clear modes. (See density plot below.)

```
meanshift500 <- read.table("meanshift500.txt", quote="\"", comment.char="")
x_small <- meanshift500$V1
h <- 0.14
d <- density(x_small, bw = h)
plot(d)
```

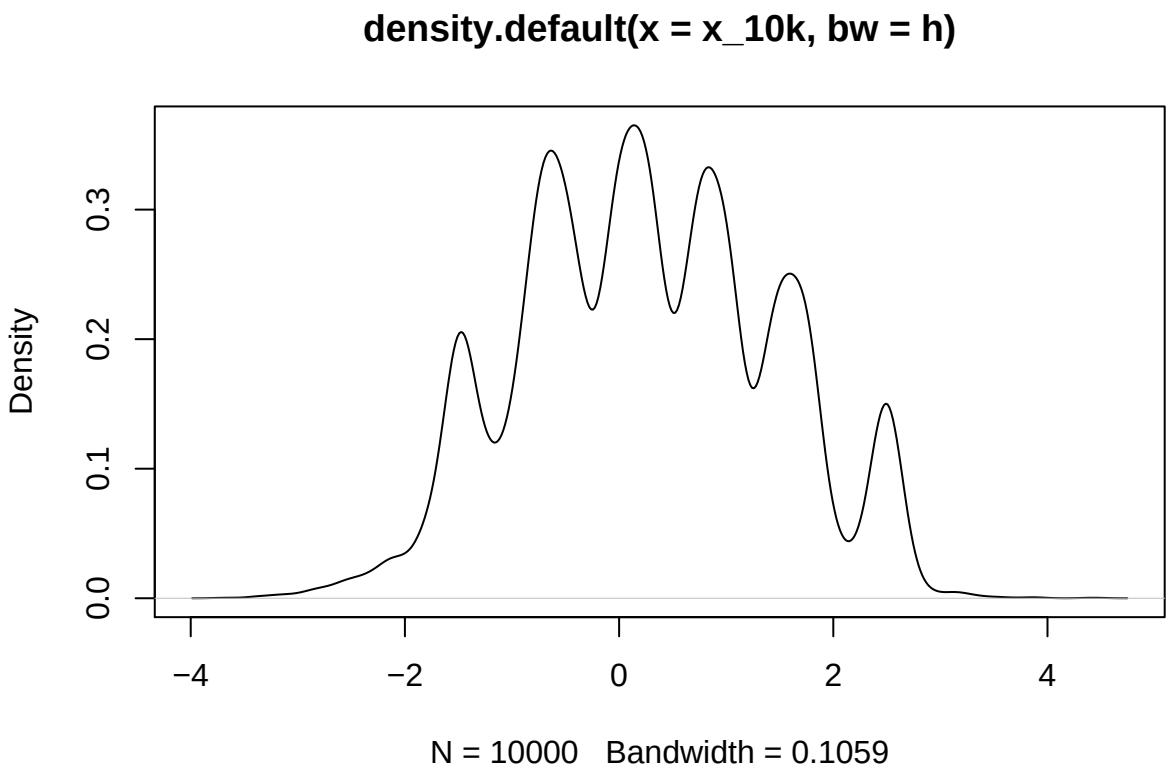



```
modes_test <- c()
for(i in seq(min(x_small), max(x_small), length = 100) ) {
  x_small_mode <- est_mode(x_small, h, start_point = i)
  modes_test <- c(modes_test, x_small_mode)
}
print(unique(round(modes_test, 3))) #Our modes for the 500
```

```
## [1] -1.512 -0.614  0.253  0.853  1.578  2.489
```

And now on the 10K size set. We use $h = 0.1059$ with similar reasoning for the 500 data.

```
meanshift10k <- read.table("meanshift10k.txt", quote="\"", comment.char="")
x_10k <- meanshift10k$V1
h <- 0.1059
d <- density(x_10k, bw = h)
plot(d)
```




```
modes_test <- c()
for(i in seq(min(x_10k), max(x_10k), length = 100) ) {
  x_mode <- est_mode(x_10k, h, start_point = i)
  modes_test <- c(modes_test, x_mode)
}
print(unique(round(modes_test, 3))) #Our modes for the 10k
```

```
## [1] -1.474 -0.637  0.140  0.836  1.594  2.492  3.118  3.857  4.427
```

2) Wizard of Ozone

a)

```
#install.packages("fields")
library(fields)
```

```
## Loading required package: spam
```

```
## Loading required package: grid
```

```
## Spam version 1.4-0 (2016-08-29) is loaded.
## Type 'help( Spam)' or 'demo( spam)' for a short introduction
## and overview of this package.
## Help for individual functions is also obtained by adding the
## suffix '.spam' to the function name, e.g. 'help( chol.spam)'.
```

```
##
## Attaching package: 'spam'
```

```
## The following objects are masked from 'package:base':
```

```
##
##     backsolve, forwardsolve
```

```
## Loading required package: maps
```

```
data("ozone2")
```

i)

The grid:

```
x <- seq(-93,-82,.1)
y <- seq(40, 46 ,.1)
```

ii)

2d kernel smoother

```
#removes NA
clean <- function(X, Y) {
  X.clean <- X[!is.na(Y),]
  Y.clean <- Y[!is.na(Y)]
  clean_set <- cbind(X.clean, Y.clean)
  colnames(clean_set) <- c("lon", "lat", "ozone")
  return(data.frame(clean_set))
}

#x, tuple (x, y) to estimate at
#Y - sample ozone concentrations
r_hat_xy <- function(lon.lat, x, Y, h){
  data <- clean(lon.lat, Y)
  lon.lat.clean <- data[,1:2]
  Y.clean <- data[,3]

  weights <- dnorm((x[1] - lon.lat.clean[,1]) / h) * dnorm((x[2] - lon.lat.clean[,2]) / h)
  weighted_val <- weights %*% Y.clean
  total_weights <- sum(weights)
  return(weighted_val / total_weights)
}
#r_hat_xy(X, c(-87, 43), Y, 0.05) #quick test
```

Function to estimate at the grid points.

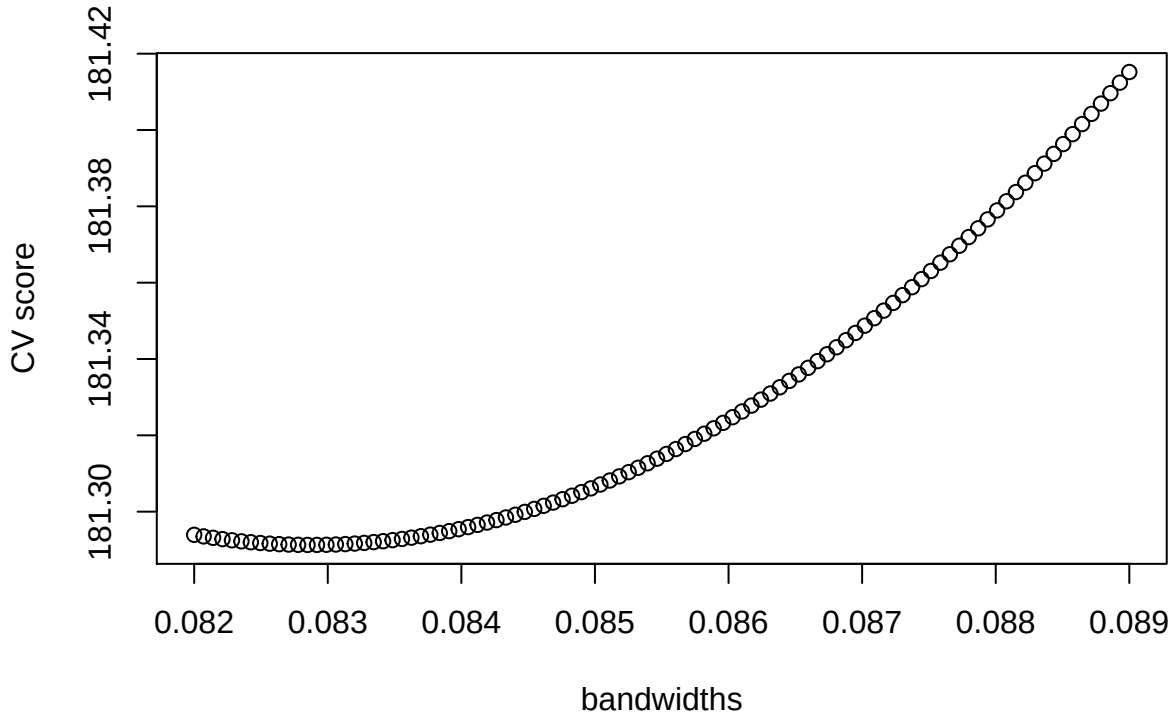
```
#X - observed data. h - bandwidth. x, y - grid points to estimate (vectors)
get_ozone_estimates <- function(X, h, Y, x, y) {
  grid_estimates <- matrix(rep(0, times = length(x) * length(y) ), nrow = length(x), byrow = TRUE )
  for(i in 1:length(x)){
    for(j in 1:length(y)){
      grid_estimates[i,j] <- r_hat_xy(X, c(x[i], y[j]), Y, h)
    }
  }
  return(grid_estimates)
}
h <- 0.05
#We want the lon.lat data for June 18th, 1987
day_index <- match("870618", ozone2$dates)
```

```
Y <- ozone2$y[day_index, ]
X <- ozone2$lon.lat
#grid_estimates <- get_ozone_estimates(X, h, Y, X[,1], X[,2] ) #test
#grid_estimates <- get_ozone_estimates(X, h, Y, x, y) #test
#grid_estimates
```

iii)

Cross Validation. L2 loss with LOO:

```
#caluclates loocv score for bandiwdith h
#X and Y must have same num of rows
#x and y are points to estimate on
loocv_2d <- function(X, Y, h) {
  l2 <- c()
  data <- clean(X, Y)
  X <- data[,1:2]
  Y <- data[,3]
  #print(dim(X))
  for(i in 1:length(Y)){
    X_new <- X[-i,]
    Y_new <- Y[-i]
    x_new <- X[i, 1]
    y_new <- X[i, 2]
    grid_est <- get_ozone_estimates(X_new, h, Y_new, x_new, y_new)
    l2 <-c(l2, Y[i] - grid_est)
  }
  return(mean (l2^2) )
}

#Now find the best bandwidth under LOOCV
h_list <- seq(0.082, 0.089, length = 100)
scores <- c()
for(h in h_list){
  scores <- c(scores, loocv_2d(X, Y, h))
}

#function to find best bandwidth
best_h <- function(cv_scores, h) {
  cv_min <- min(cv_scores)
  h_min_index <- match(cv_min, cv_scores)
  h_min <- h[h_min_index]
  return(h_min)
}
plot(scores ~ h_list, xlab = "bandwidths", ylab = "CV score")
```




```
print(h <- best_h(scores, h_list))
```

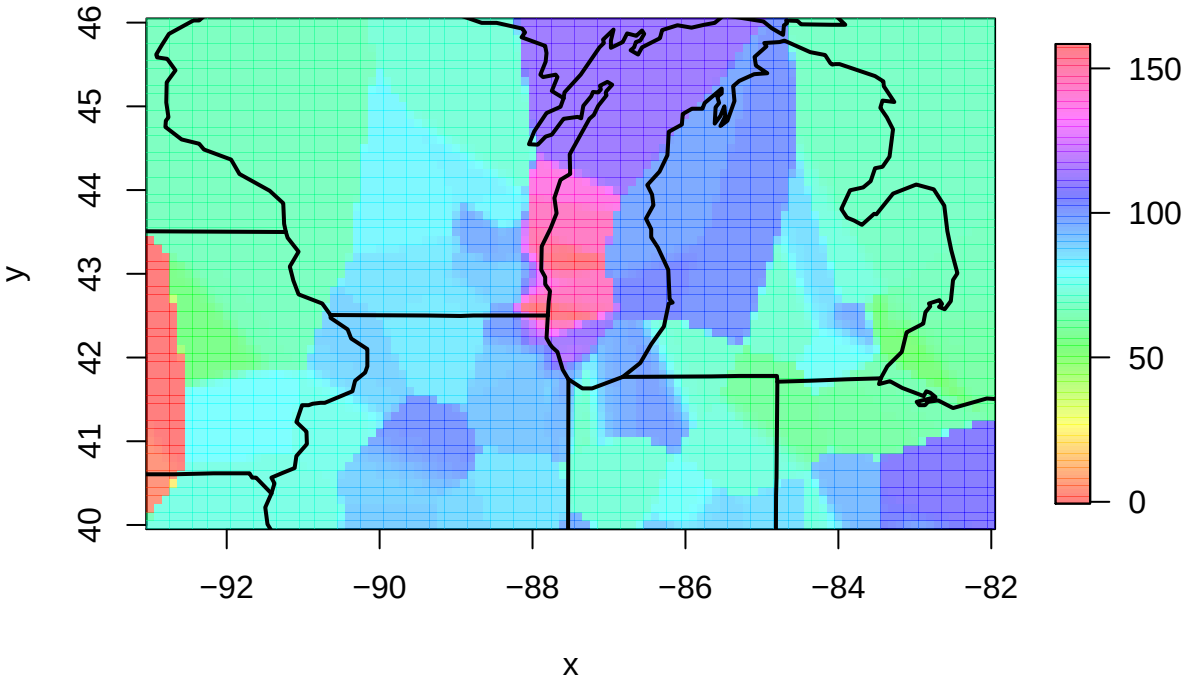
```
## [1] 0.08284848
```

This we will use the bandwidth of 0.08284848, because that bandwidth will lead to the lowest cross validation score.

iv)

Heatmap

```
z <- get_ozone_estimates(X, h, Y, x, y)
image.plot(x, y, z, col=rainbow(128,alpha=.5))
US(add=T, lwd=2, col=1)
```



b)

Iterative Updating Rule

4) Boostrap Estimation

Constants to be used throughout q4):

```
CI = function(th,th.hat,n,alpha) {
  #r = sqrt(n)*(th-th.hat)
  #left  = th.hat - quantile(r,1-alpha/2)/sqrt(n)
  #right = th.hat - quantile(r,alpha/2)/sqrt(n)
  se_boot <- sqrt(var(th))
  left <- th.hat + qnorm(p = alpha/2) * se_boot
  right <- th.hat + qnorm(p = 1 - alpha/2) * se_boot
  return(c(left,right))
}

B <- 100 #num of boostrap samples
N <- 50 #number of experiments

n <- 100 #num of (actual) samples to generate
alpha = .05
left = rep(0,N)
right = rep(0,N)
```

a)

```
B_0 <- -1
B_1 <- 2
B_2 <- -1
true.theta <- -B_1/(2 * B_2)

for(i in 1:N) {
  x <- runif(n, max = 2)
  e <- rnorm(n, sd = 0.2)
  y <- B_0 + B_1 * x + B_2 * x^2 + e

  l_mdl <- lm(y ~ x + I(x^2) )
  B_2_hat <- l_mdl$coefficients[3]
  B_1_hat <- l_mdl$coefficients[2]
  th.hat <- -B_1_hat/(2 * B_2_hat)

  th <- c()
  for(j in 1:B) {
    bs_indices <- sample(1:length(x), replace = TRUE)
    x_star <- x[bs_indices]
    y_star <- y[bs_indices]

    l_mdl_bs <- lm(y_star ~ x_star + I(x_star^2) )

    B_2_hat_bs <- l_mdl_bs$coefficients[3]
    B_1_hat_bs <- l_mdl_bs$coefficients[2]
    max_hat_bs <- -B_1_hat_bs/(2 * B_2_hat_bs)
    th <- c(th, max_hat_bs)
  }
  interval = CI(th,th.hat,n,alpha)
  left[i]  = interval[1]
  right[i] = interval[2]
}

coverage = mean((left <= true.theta)&(right >= true.theta))
cat("Coverage = ",coverage,"\n")
```

```
## Coverage =  0.96
```

b)

```
#first let's find the true theta, true.theta
var_x <- 101
var_y <- 101
var_z <- 1
cov_xy <- 100
cov_xz <- 10
cov_yz <- 10

c_m <- matrix(c(var_x, cov_xy, cov_xz,
                cov_xy, var_y, cov_yz,
                cov_xz, cov_yz, var_z), nrow = 3)
omega <- matrix(c(1, 0, -10,
                  0, 1, -10,
```

```
                  -10, -10, 201), nrow = 3, byrow = 3) #inverse of covariance matrix

true.theta <- -omega[1,2] / sqrt(omega[1,1] * omega[2,2]) #zero

for(i in 1:N) {
  z <- rnorm(n)
  x <- 10*z + rnorm(n)
  y <- 10*z + rnorm(n)

  X <- matrix(c(x, y, z), ncol = 3)
  cov_X_hat <- cov(X)
  omega_hat <- solve(cov_X_hat)
  th.hat <- -omega_hat[1,2] / sqrt(omega_hat[1,1] * omega_hat[2,2])

  th <- c()
  for(j in 1:B) {
    bs_indices <- sample(1:length(z), replace = TRUE)
    x_star <- x[bs_indices]
    y_star <- y[bs_indices]
    z_star <- z[bs_indices]

    X_star <- matrix(c(x_star, y_star, z_star), ncol = 3)
    cov_X_hat_star <- cov(X_star)
    omega_hat_star <- solve(cov_X_hat_star)
    th.hat.bs <- -omega_hat_star[1,2] / sqrt(omega_hat_star[1,1] * omega_hat_star[2,2])
    th <- c(th, th.hat.bs)
  }
  interval = CI(th,th.hat,n,alpha)
  left[i]  = interval[1]
  right[i] = interval[2]
}

coverage = mean((left <= true.theta)&(right >= true.theta))
cat("Coverage = ",coverage,"\n")
```

```
## Coverage =  0.96
```

c)

```
#install.packages("MASS")
library(MASS)
I <- matrix(rep(0, times = 100), nrow = 10)
for(i in 1:10){
  I[i, i] = 1
}
true.theta <- min(eigen(I)$values)

for(i in 1:N) {
  X <- mvrnorm(n, rep(0, 10), I)
  cov_hat <- cov(X)
  th.hat <- min(eigen(cov_hat)$values)
```

```
  th <- c()
  for(j in 1:B) {
    bs_indices <- sample(1:nrow(X), replace = TRUE)
    X_star <- X[bs_indices,]
    cov_bs <- cov(X_star)

    theta_bs <- min(eigen(cov_bs)$values)
    th <- c(th, max_hat_bs)
  }
  interval = CI(th,th.hat,n,alpha)
  left[i]  = interval[1]
  right[i] = interval[2]
}

coverage = mean((left <= true.theta)&(right >= true.theta))
cat("Coverage = ",coverage,"\n")
```

```
## Coverage =  0
```

The coverage is 0 becasuse the sample eigenvalues are def going to have values below 1.

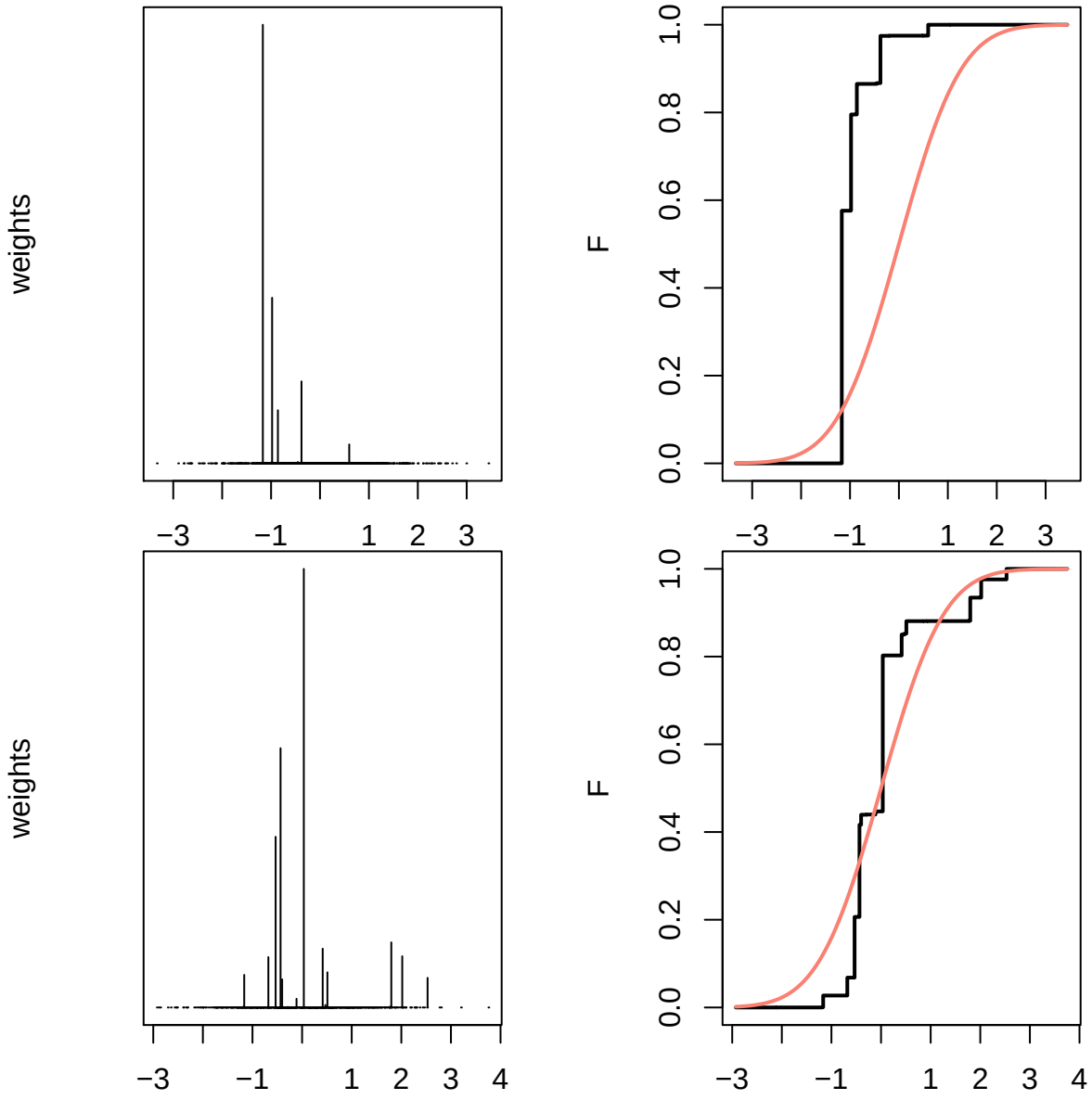
5)

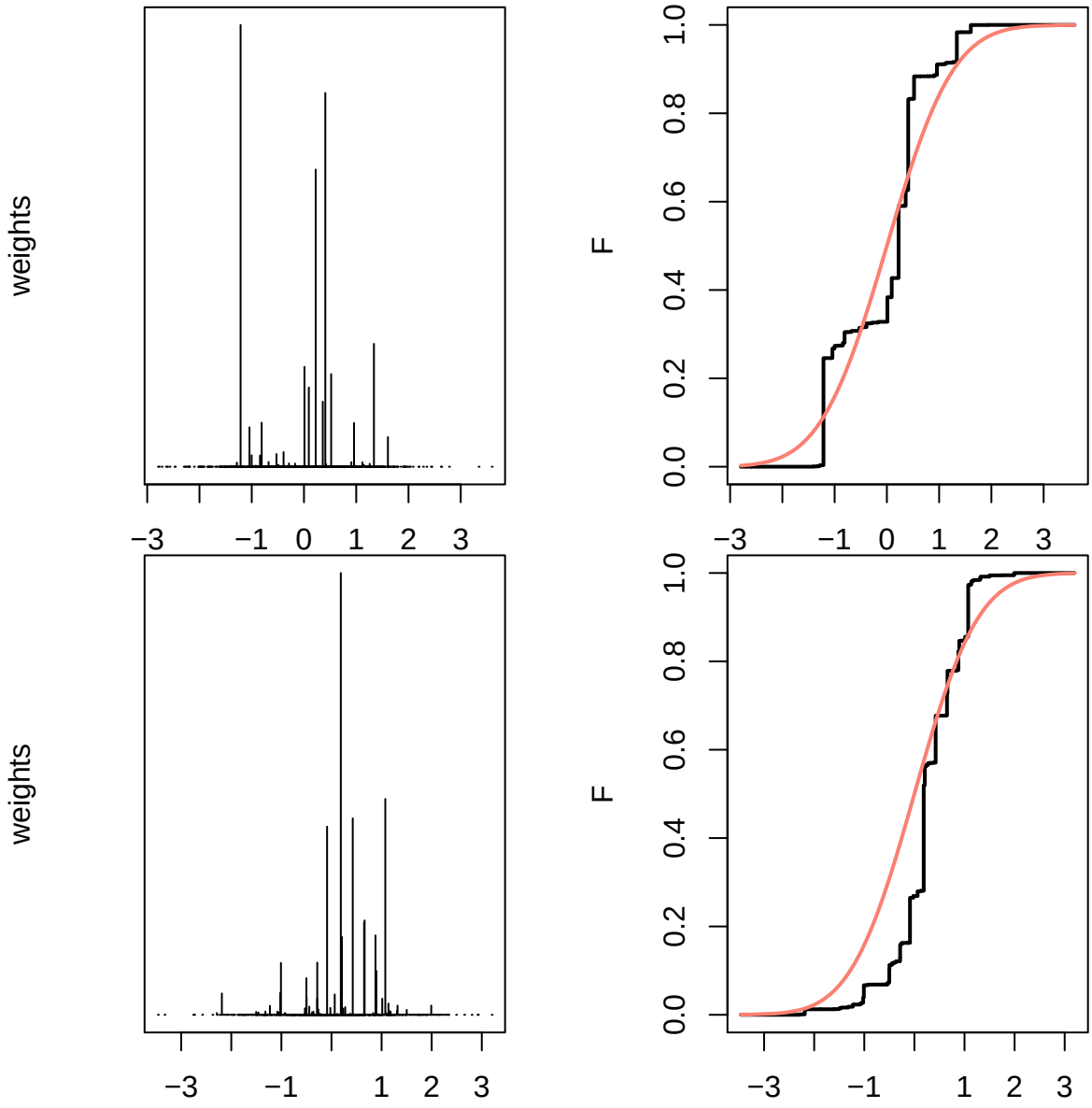
d)

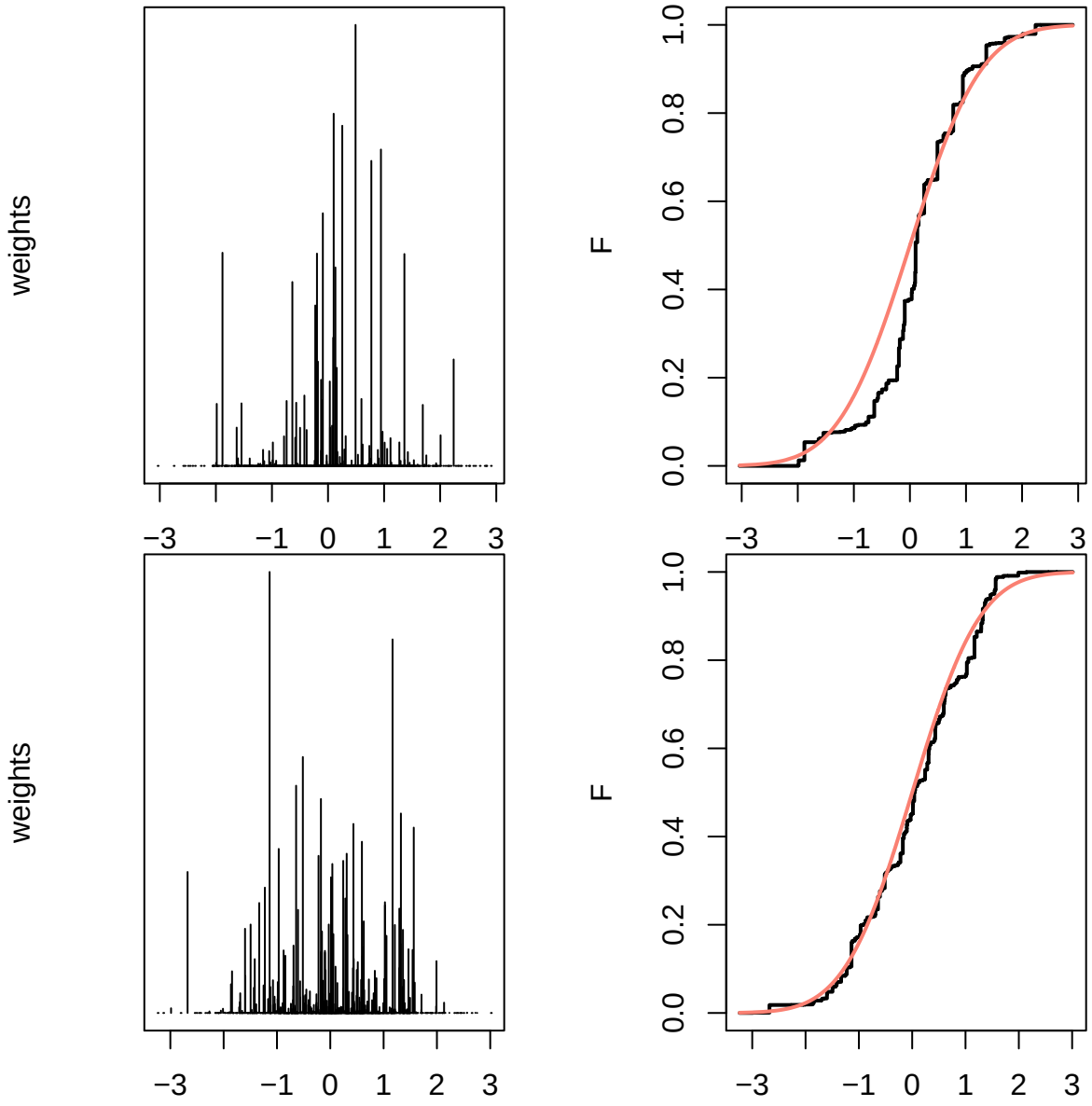
Functions to define stick breaking process (to generate weights) and sample diritch process.
Credits to Le Prof Lafferty

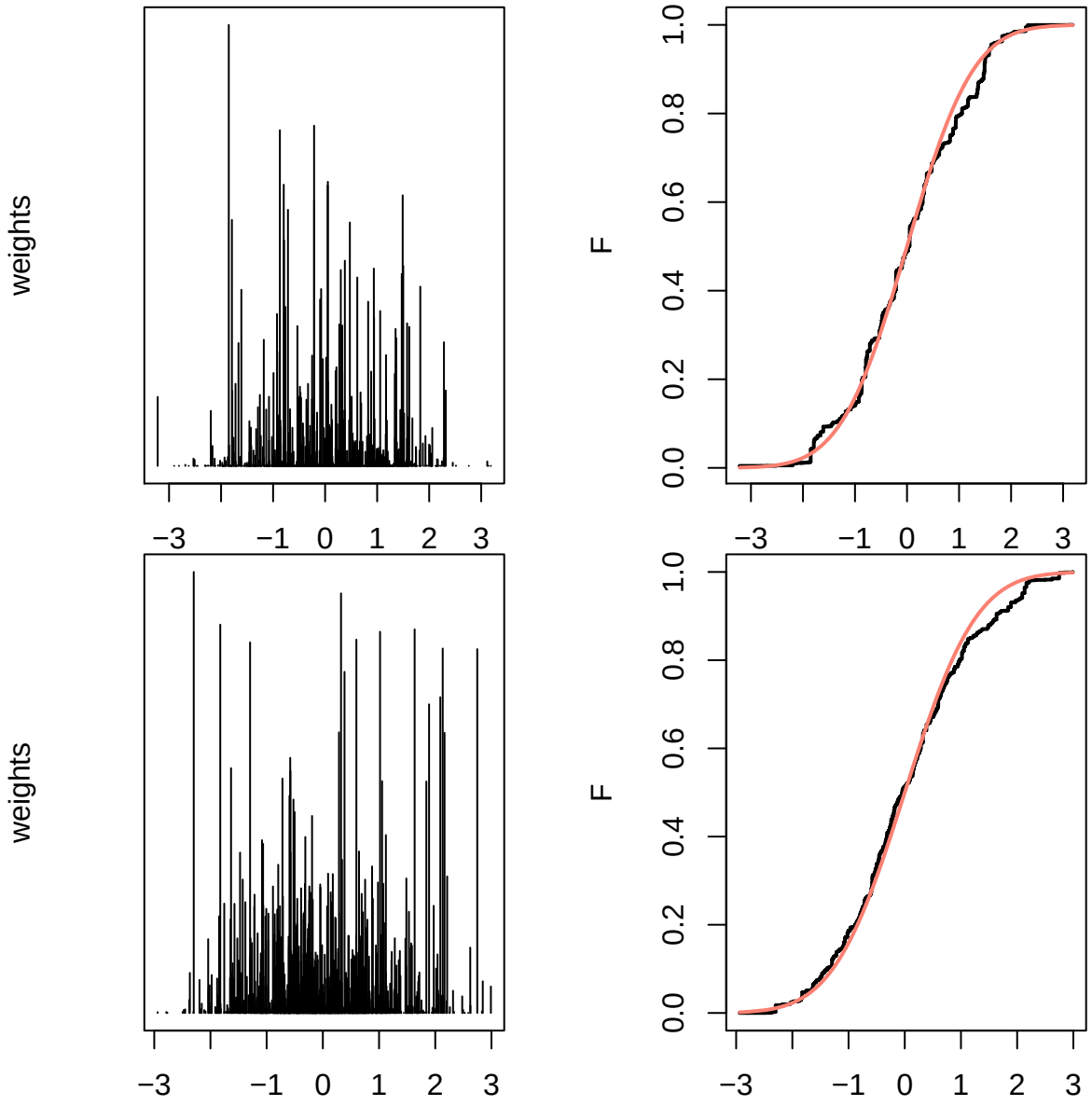
```
stick.break = function(alpha,N){
  V = rbeta(N,1,alpha)
  w = V
  d = 1
  for(i in 2:N) {
    d = d*(1-V[i-1])
    w[i] = V[i]*d
  }
  return(w/sum(w))
}

sample.dirichlet.process = function(alpha,mu = 0,sigma = 1, obs = FALSE, use_f_bar= FALSE) {
  N = 1000
  if(use_f_bar){
    s = generate_F_bar(num = N, n = length(obs), alpha = alpha, obs = obs)
  }
  else{
    s = rnorm(N,mu,sigma)
  }

  w = stick.break(alpha,N)
  o = order(s)
  s = s[o]
  w = w[o]
  F = cumsum(w)
```

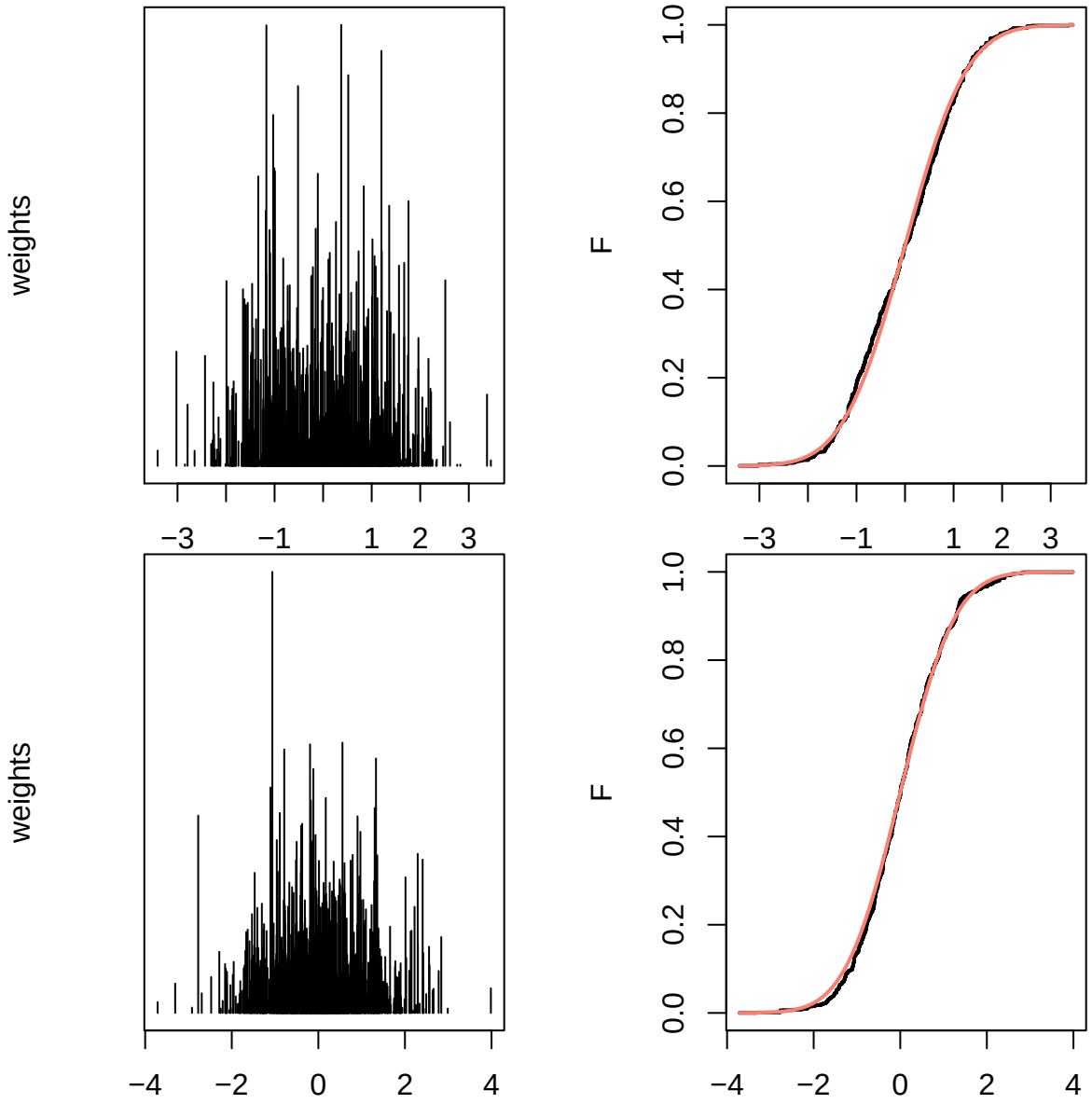
```
  if(!use_f_bar){
    par(mfrow=c(1,2))
    plot(s,w,type="h",xlab="",ylab="weights",yaxt="n")
    plot(s,F,type="s",xlab="",ylab="F",lwd=2)
    lines(s,pnorm(s,mean=mu,sd=sigma),lwd=2,col="salmon")
  }

  else{
  dp_post <- matrix(c(s, F), ncol = 2)
  colnames(dp_post) <- c("x", "F")
  return(data.frame(dp_post))
  }
}

alphas <- c(1, 2, 5, 10, 20, 50, 100, 200, 500, 1000)
for(alpha in alphas) {
  sample.dirichlet.process(alpha=alpha)
}
```


weights
−3
−1
1
2
3
F
0.0
0.2
0.4
0.6
0.8
1.0
weights
−3
−1
1
2
3
4
F
0.0
0.2
0.4
0.6
0.8
1.0
−3
−1
1
2
3
4


weights
F
weights
F
−3
−1
0
1
2
3
0.0
0.2
0.4
0.6
0.8
1.0


weights
F
weights
F
−3
−1
0
1
2
3
0.0
0.2
0.4
0.6
0.8
1.0


weights
weights
F
F
−3
−1
0
1
2
3
0.0
0.2
0.4
0.6
0.8
1.0



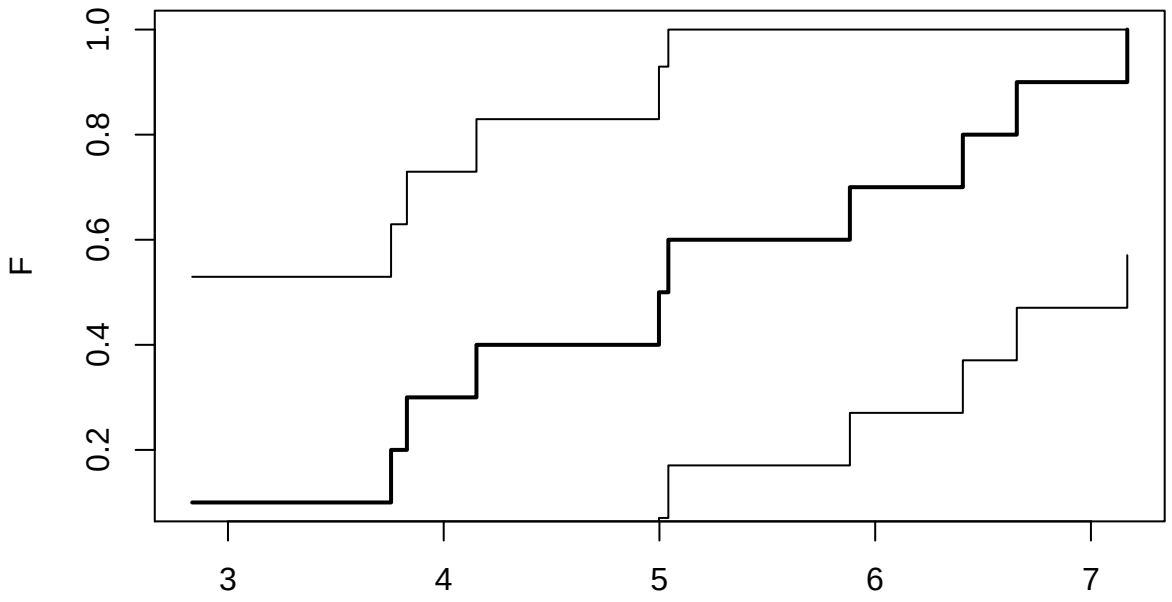
e)

i) Empirical CDF CI

Using DKW, we know the value of epsilon (in Wasserman)

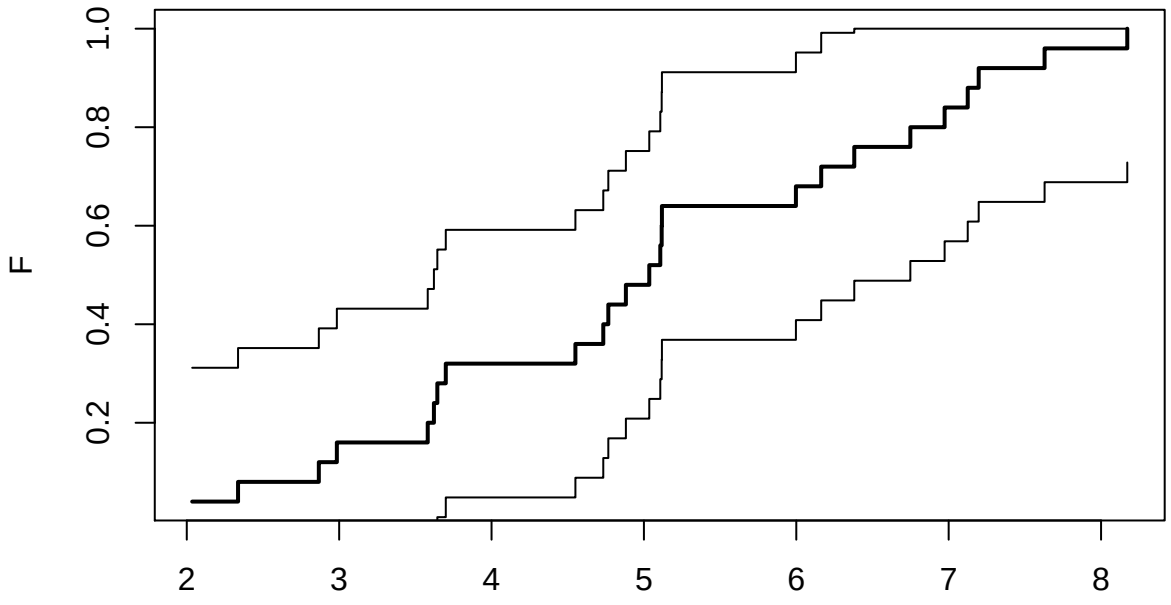
```
par(mfrow = c(1,1))

#Confidence Bands, x is point to est the lower and upper bounds at
cdf_bounds <- function(n, x, F_hat, alpha = 0.05){
  epsilon <- sqrt(1/(2*n) * log(2 / alpha))
  lower <- c()
  upper <- c()
  for(i in x) {
    lower <- c(lower, max( c(F_hat(i) - epsilon, 0) ) )
    upper <- c(upper, min( c(F_hat(i) + epsilon, 1) ) )
```

```
  }
  return( matrix(c(lower, upper), ncol = 2 ) )
}

plot_ecdf_bands <- function(n){
  x <- rnorm(n, mean = 5, sd = sqrt(3))
  x_star <- x[order(x)]
  F_hat <- ecdf(x_star)

  plot(x_star,F_hat(x_star),type="s",xlab="",ylab="F",lwd=2, main = cat(n, "samples"))
  n <- length(x)
  a <- cdf_bounds(n, x_star, F_hat)
  lines(x_star, a[,1], type = "s")
  lines(x_star, a[,2], type = "s")
}

plot_ecdf_bands(n = 10)
```



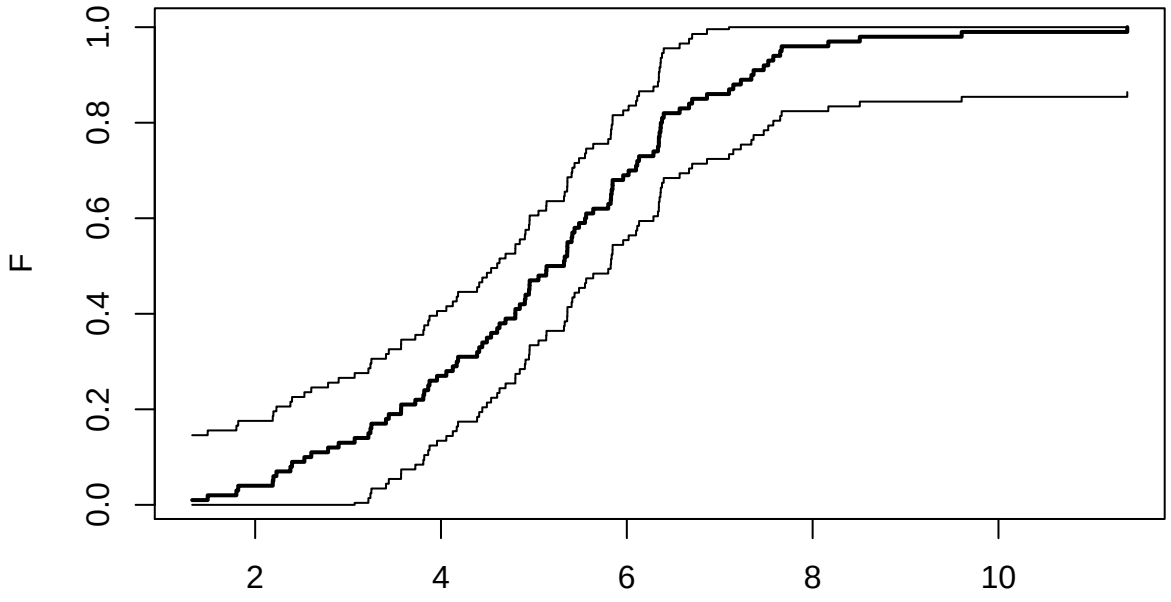

```
## 10 samples
```

```
plot_ecdf_bands(n = 25)
```




```
## 25 samples
```

```
plot_ecdf_bands(n = 100)
```



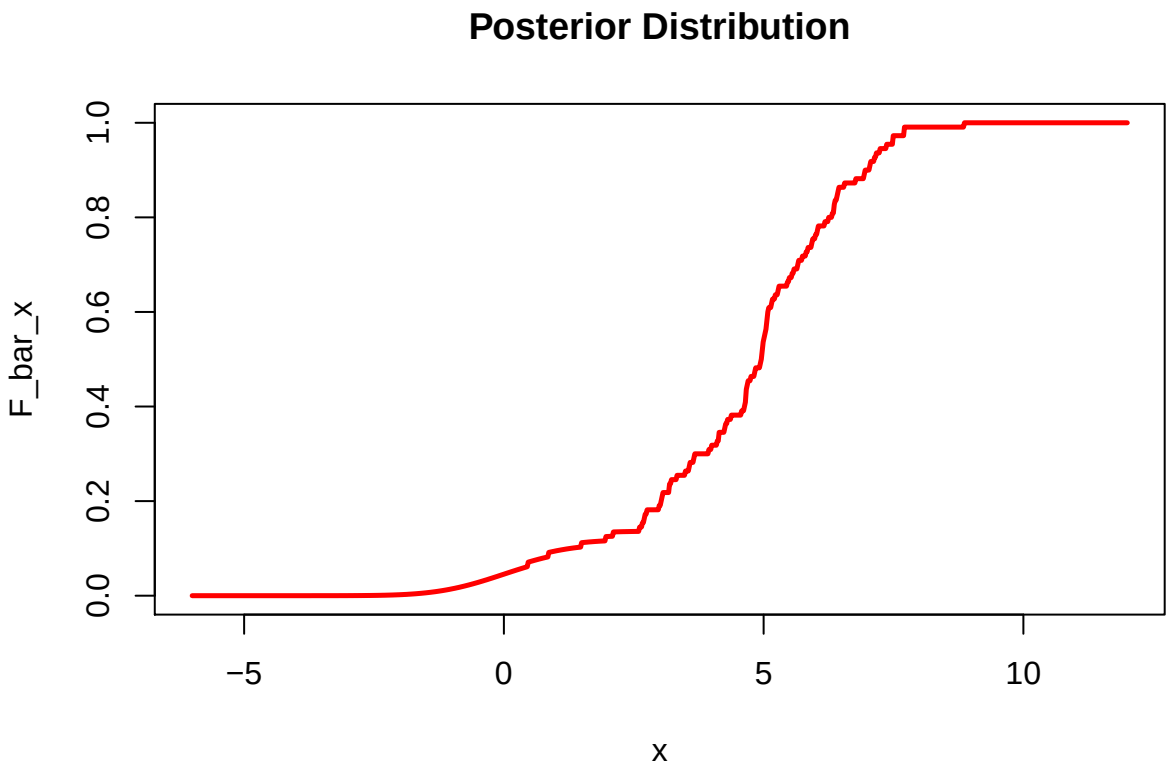

```
## 100 samples
```

ii)

The Bayesian posterier is plotted below

```
n <- 100
obs <- rnorm(n, mean = 5, sd = sqrt(3))
F_hat <- ecdf(obs)

#assumes F_0 to be N(0,1). Returns probabillity p, where F_bar(x) = p
F_bar <- function(x, F_hat, alpha = 10, n = 100) {
  return(n/(alpha + n) * F_hat(x) + alpha/(n + alpha) * pnorm(x))
```

```
}
#F_bar(2, 10, n, F_hat) #test to see if it works

x <- seq(-6, 12, length = 1000)
F_x <- c()
for(i in x){
  F_x <- c(F_x, F_bar(i, F_hat) )
}
par(mfrow = c(1,1))
plot(F_x ~ x, type = "l", xlab = "x", ylab = "F_bar_x", main = "Posterior Distribution", col = "red", lw
```




```
#assumes N(0,1) base distribution. Num is number of points to generate
generate_F_bar <- function(num, n, alpha, obs){
  bound <- alpha/(n + alpha)
  samples <- c()
  for(i in 1:num){
    check <- runif(1)
    if(check <= bound){
      samples <- c(samples, rnorm(1)) #take from N(0,1)
    }
    else{
      samples <- c(samples, sample(obs, size = 1, replace = TRUE))
    }
  }
  return(samples)
}

#generate_F_bar(10, n = 100, alpha = 10, obs = obs) test
dp_dists <- c()
```

```
#for(i in 1:10){
#  dp <- sample.dirichlet.process(alpha = 10, obs = obs, use_f_bar = TRUE)
#  lines(dp$F ~ dp$x, type = "s", col = "blue", lwd = 0.2)
#}
```